

Running Jobs on IBM SP

San Diego Supercomputing Center



San DIEGO SUPERCOMPUTER CENTER

Overview

- System overview and how CPUs are requested
- Compiling
- Running parallel jobs
 - Interactive
 - Loadleveler scripts



Overview and CPUs are Requested

- Blue Horizon has 144 Nodes with 8 CPUs on each node. $144 \times 8 = 1152$ CPUs
- Habu has 334 Nodes with 4 CPUs on each node. $334 \times 4 = 1336$ CPUs
- Request:
 - # of Nodes
 - # of tasks per Node
 - # of threads per task (hybrid code only)
- # of CPUs used = nodes*tasks*threads
- Limitations of MPI (4 tasks per node)
- Hybrid code spawns threads per task



NPACI Environment: Logging In

- `ssh horizon.npaci.edu -l username`
- `ssh` (secure shell) required for secure access
- see: <http://security.sdsc.edu/>



NAVO Environment Logging In

Access via kerberos or ssh:

- ktelnet -x -F habu.navo.hpc.mil
- krlogin -x -F habu.navo.hpc.mil
- krsh -x -F habu # short names allowed inside the NAVO MSRC network
- ssh habu # ssh that supports Kerberos tickets
- kftp habu



NPACI IBM SP Filesystem Structure

- \$HOME / : < 50 MB/user of backed up storage used for storage of unix essentials, etc.
- GPFS available
- \$WORK / : 255/36 GB of purged (96 hours) disk storage primary workspace
- Archival Storage: HPSS
 - Disk/tape system providing TBytes of storage



NAVO IBM SP Filesystem Structure

GPFS filesystem /scr (2.92TB) - used for temporary data storage and for storage during batch processing.

- /u/home : 512 MB/user for user \$HOME directory
- Users are responsible for archiving files from /scr and /u/home - no system backups of /u/home, /scr or /scratch filesystems (local to each node) will be made by MSRC staff.
- system "scrubber" will monitor utilization of /scr space.
- usr/local/bin/quota command gives quota usage and limits in kilobyte units.



Compiling C

- C
 - xlc program.c
- C MPI
 - mpcc program.c
- C OpenMP
 - cc_r -qsmp=noauto:omp program.c
- C MPI and OpenMP
 - mpcc_r -qsmp=noauto:omp program.c
- C++
 - xlc mpCC xlc_r mpCC_r



Compiling Fortran

- Fortran
 - `xlf program.f`
- Fortran MPI
 - `mpxlf program.f`
- Fortran OpenMP
 - `xlf_r -qsmp=noauto:omp program.f`
- Fortran MPI and OpenMP
 - `mxlf_r program -qsmp=noauto:omp program.f`
- Fortran 90
 - `xlf90, mpxlf90 xlf90_r mpxlf90_r`



Useful Optimizations

- -O3 or -O4 “good level of optimization”
- -qstrict “assures program semantics”
- -qarch=auto or -qarch=pwr3
- -qtune=auto or -qtune=auto
- **-For C and C++ try the following**
 - -qnoipa (safe may or may not help)
 - -qalias=allpt (make sure there is no pointer aliasing)



Running on IBM SP - PE

IBM Parallel Environment (PE) is primary support for those developing parallel applications on SP

- OS support
- Compilers, debugger, profiler (F90/C/C++)
- Message-passing libraries (MPI, LAPI)
- Numerical libraries (ESSL/PESSL)
- Batch scheduler - LoadLeveler
- Parallel Operating Environment (POE)
 - Primary user interface to PE
 - Starting parallel processes
 - Many environment variables



Running Interactive Jobs

- Interactive Jobs:

```
poe a.out arg_list -nodes n -tasks_per_node m  
-rmpool 1
```

tasks_per_node: 1-n (number of CPUs on each node)

If running OpenMP set OMP_NUM_THREADS environmental variable



Batch Jobs with LoadLeveler

LoadLeveler - utility that incorporates batch processing support and job scheduler and is used on many IBMs to help maximize processing throughput.

- LoadLeveler offers command line interface
 - llsubmit, llq, llcancel, etc.
- also X-Windows based GUI - xloadl.
- LoadLeveler can be “customized” to match resources and use policies at a site.



Batch Jobs with LoadLeveler

- LoadLeveler “job scripts” submitted to batch queues and then released to execute as system resources become available. The scheduler tries to make the best use of resources by:
 - filling the available processors/nodes



Batch Jobs with LoadLeveler

- Develop a program and make sure it runs correctly
- Write a script file that has information about your job and the nodes/tasks/threads you want to run it on
- Submit the script file to the LoadLeveler
- Check the status of your job
- NPACI Batch Script Generator (sample)

<http://hotpage.npaci.edu/Batch>



Writing LoadLeveler Scripts

- arguments* - arguments to pass to the executable
- input* - file to use as *stdin*
- output* - file to use as *stdout*
- initialdir* - initial working directory for job
- executable* - executable or script to run
- job_type* - can be parallel or serial
- notify_user* - user to send email to regarding this job, keywords are case insensitive



Keywords for LoadLeveler Scripts (cont.)

- | | |
|-------------------------|--|
| <i>notification</i> | - e-mail to notify_user
(always, error, start, never, complete) |
| <i>node</i> | - how many nodes do you want |
| <i>tasks_per_node</i> | - MPI tasks/node (max of 4 for now) |
| <i>wall_clock_limit</i> | - maximum wall clock time to use for the job |
| <i>class</i> | - class (queue) to run in
(low,normal,high,express) |
| <i>queue</i> | - puts the job in the queue |



Keywords for LoadLeveler Scripts (cont.)

NAVO SP LL scripts - following LL elements are required:

#@ node = ###	(### = integer value)
#@ tasks_per_node = ###	(### = integer value)
#	(or #@ total_tasks = ###)
#@ wall_clock_limit = h:m:s	(hms = hours:minutes:seconds)
#@ class = classname	(batch is default class for users)
#@ job_type = xxxxx	(parallel or serial)
#@ account_no = aaaaaa	(Mandatory)
#@ node_usage = not_shared	(Only option accepted)
#@ queue	(requires no additional argument)



LoadLeveler Scripts: Example 1

Normal MPI job with fast interconnect (us) so only 4 CPUs per node

```
#!/bin/ksh
# @ environment = MP_EUILIB=us; MP_SHARED_MEMORY=YES; MP_PULSE=0;
#   MP_INTRDELAY=100;
# @ job_name = scal
# @ class = high
# @ job_type = parallel
# @ node=16
# @ tasks_per_node=4
# @ node_usage=not_shared
# @ network.MPI=css0,not_shared,US
# @ wall_clock_limit =3:30:00
# @ input = /dev/null
# @ output = /work/login-name/mydir/my.out
# @ error = /work/login-name/mydir/my.err
# @ initialdir = /work/login-name/mydir
# @ executable = myscript
# @ notify_user = login-name@npaci.edu
# @ notification = always
# @ queue
```

myscript:

```
#!/bin/ksh
```

```
a.out
```



LoadLeveler Scripts: Example 2

MPI job with fast interconnect (us), 4 CPUs per node PLUS 2 threads per task

```
#!/bin/ksh
# @ environment = MP_EUILIB=us; MP_SHARED_MEMORY=YES; MP_PULSE=0;
#   MP_INTRDELAY=100;
# @ job_name = scal
# @ class = high
# @ job_type = parallel
# @ node=16
# @ tasks_per_node=4
# @ node_usage=not_shared
# @ network.MPI=css0,not_shared,US
# @ wall_clock_limit =3:30:00
# @ input = /dev/null
# @ output = /work/login-name/mydir/my.out
# @ error = /work/login-name/mydir/my.err
# @ initialdir = /work/login-name/mydir
# @ executable = myscript
# @ notify_user = login-name@npaci.edu
# @ notification = always
# @ queue
```

myscript:

```
#!/bin/ksh

export OMP_NUM_THREADS=2
export SPINS=0
export YIELDS=0
export SPINLOOPTIME=5000

a.out
```



LoadLeveler Commands

- Job Submission

llsubmit filename submit job filename to LoadLeveler

llcancel [-q] jobid cancel a job with the given id

llq [-s job_number] list job [job number]

- Job Status

showq show job queue

showbf show backfill



IBM SP Documentation

General IBM SP Documentation

http://www.rs6000.ibm.com/resource/aix_resource/sp_books

NAVO SP Documentation

<http://www.navo.hpc.mil/usersupport/IBM>

NPACI Blue Horizon Documentation

<http://www.npaci.edu/Horizon>



Lab example

- **Log into SP login node**

```
ssh -l username bluehorizon.npaci.edu  
ssh -l username habu.navo.hpc.mil
```

- **So we all have a consistent environment (NPACI only)**

```
/usr/local/paci/shellrc/bin/COPYDEFAULTS
```

- **Copy sample LoadLeveler scripts**

```
cp /work/training/LoadLeveler/* .
```

- **Edit LLex1, LLex2, LLex3, putting in your user name, directory, etc.**

- **Compile sample code:**

```
mpxlf_r -qsmp=noauto -bmaxdata:0x20000000 ex_code.f
```

- **Run code interactively:**

```
MPI ONLY: poe a.out -nodes 1 -tasks_per_node 4 -rmpool 1
```

```
MPI/OpenMP:
```

```
setenv OMP_NUM_THREADS 2
```

```
poe a.out -nodes 1 -tasks_per_node 4 -rmpool 1
```

- **Run in batch mode:** (compare times with 3 batch examples and 2 interactive examples)

```
llsubmit LLex1
```

```
llsubmit LLex2
```

```
llsubmit LLex3
```



Lab example

```
include 'mpif.h'

real*8, allocatable, dimension(:, :) :: c
real*8, allocatable, dimension(:) :: b
real*8 sum, final_sum
real*8 total, total_time
integer hz, tstart, tstop, i, j
integer npe, iam, ierr, size, error, terr

! set up MPI
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,npe,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,iam,ierr)

! allocate local arrays where global size is 24000
size = 24000/npe
allocate( c(size,size),b(size),STAT=error)

! check that enough memory for allocation
call MPI_ALLREDUCE(error,terr,1,MPI_INTEGER,MPI_SUM,
.           MPI_COMM_WORLD,ierr)
if( terr .ne. 0 ) then
  print *, 'not enough space'
```



Lab example (Continued)

```
goto 100
end if
! initialize variables
call setup(sum,b,c)
! start clock/timer
call system_clock(count_rate=hz)
call system_clock(count=tstart)
! start calculation
!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(i,j) REDUCTION(+:sum)
do i = 1,size
do j= 1,size
sum = sum + b(i)*c(i,j)
end do
end do
! collect sum values from all mpi processes
call MPI_REDUCE(sum,final_sum,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
.           MPI_COMM_WORLD,ierr)
```



Lab example (Continued)

```
! stop clock and calculate local time
```

```
    call system_clock(count=tstop)
```

```
    total= real(tstop-tstart)/real(hz)
```

```
! since time spent is determined by the slowest process what is max time
```

```
    call MPI_REDUCE(total,total_time,1,MPI_DOUBLE_PRECISION,MPI_MAX,0,  
    .           MPI_COMM_WORLD,ierr)
```

```
! print results
```

```
    if (iam .eq. 0 ) then
```

```
        print *, 'time = ',total_time,' final sum =',final_sum
```

```
    end if
```

```
100  continue
```

```
    call MPI_FINALIZE(ierr)
```

```
end
```

```
subroutine setup(sum,b,c,size)
```

```
integer size, i,j
```

```
real*8 sum,b(size),c(size,size)
```

```
sum = 0.0
```

```
do i = 1,6000
```



Lab example (Continued)

```
b(i) = real(i)*10.0  
      do j= 1,6000  
        c(j,i) = real( i*j)  
      end do  
    end do  
    return  
  
end
```

